

Object Detection and Distance Measurement Device

Object Detection and Distance Sensing for Visually impaired people

Made by

Madhav

Table of Contents

S.no	Title	Page no.
1	Abstract	3
2	Introduction	3
3	Literature Review	3
4	Hardware Components	4 to 5
5	Raspberry Pi 4 Model B	6 to 7
6	Raspberry Pi Camera Module 2	8 to 9
7	UPS HAT (D)	10 to 12
8	Volume Buttons	13 to 15
9	LiDAR Distance Sensor	16 to 17
10	Object Detection	18 to 20
11	Main Code	21 to 25
12	Program that automatically starts main code on bootup	26 to 27
13	Depth Estimation MiDaS Neural Network	28 to 30
14	3D Model	31 to 32
15	Conclusion	33
16	Cost	34
17	Future plans	35
18	References	35

Abstract

The aim of this project is to introduce an innovative assistive device designed to empower individuals with visual impairments. Built on the Raspberry Pi 4 platform, this device combines YOLOv8 object detection with the TF-Luna LiDAR sensor to detect indoor objects within a 1.5-to-3-meter range. The collected data is transformed into audio messages using Google Text-to-Speech (gTTS) and transmitted to an earpiece via an auxiliary (AUX) connector. This portable and user-friendly device offers real-time information and facilitates safe indoor navigation. It is documented and open source, aiming to enhance the quality of life and promote independence for the visually impaired.

Introduction

Our project aims to create a device to help people with visual impairments. We're using YOLOv8 object detection and a TF-Luna LiDAR sensor with a Raspberry Pi 4. The device's main goal is to make it easier for people to navigate indoors.

With YOLOv8, the device can recognize objects like chairs, sofas, and tables in indoor spaces. The TF-Luna LiDAR sensor measures how far these objects are from the user within a 1.5 to 3-meter range. By combining data from these two sources, the device uses Google Text-to-Speech to turn this information into spoken messages that tell users about the objects and where they are.

Users can listen to these messages through an earpiece connected to the device. We've made sure to make the device easy to use and accessible, with options like voice commands and tactile controls for different users.

Literature Review

In the world of helping blind people, there are cool gadgets like smart glasses that use AI, headphones that work through your bones, and special keyboards for Braille. But our device is different because it does more than just tell you what's around. It also tells you exactly how far away things are. This helps blind people know more about what's around them.

Other tools are great, but our device is super advanced and can make blind people more independent and safer. It helps them understand their surroundings better, so they know what's going on when they're moving around.

Hardware components:

Raspberry Pi 4: A compact, versatile computer that serves as the core platform for our project, facilitating data processing and communication.



TF-Luna LiDAR Distance Sensor: A precise LiDAR sensor designed for measuring distances, enabling the device to determine the proximity of detected objects within a range of 1.5 to 3 meters.



The Raspberry Pi Camera Module V2: It is a small, high-quality camera designed for Raspberry Pi boards. It features an 8-megapixel sensor and provides clear and vibrant images and videos. With its compact size and ease of integration, it is a popular choice for a wide range of DIY and IoT projects.



The UPS HAT (D): It is a specialized variant of the Uninterruptible Power Supply (UPS) hardware attachment designed for Raspberry Pi boards. It provides backup power during outages or voltage fluctuations, safeguarding data and ensuring continuous operation. The "(D)" signifies unique features or specifications specific to this UPS HAT model, which may include extended battery life, additional ports, or enhanced power management.

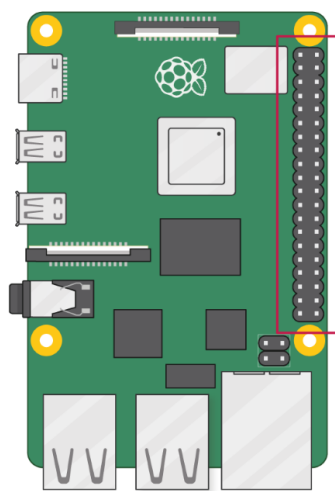
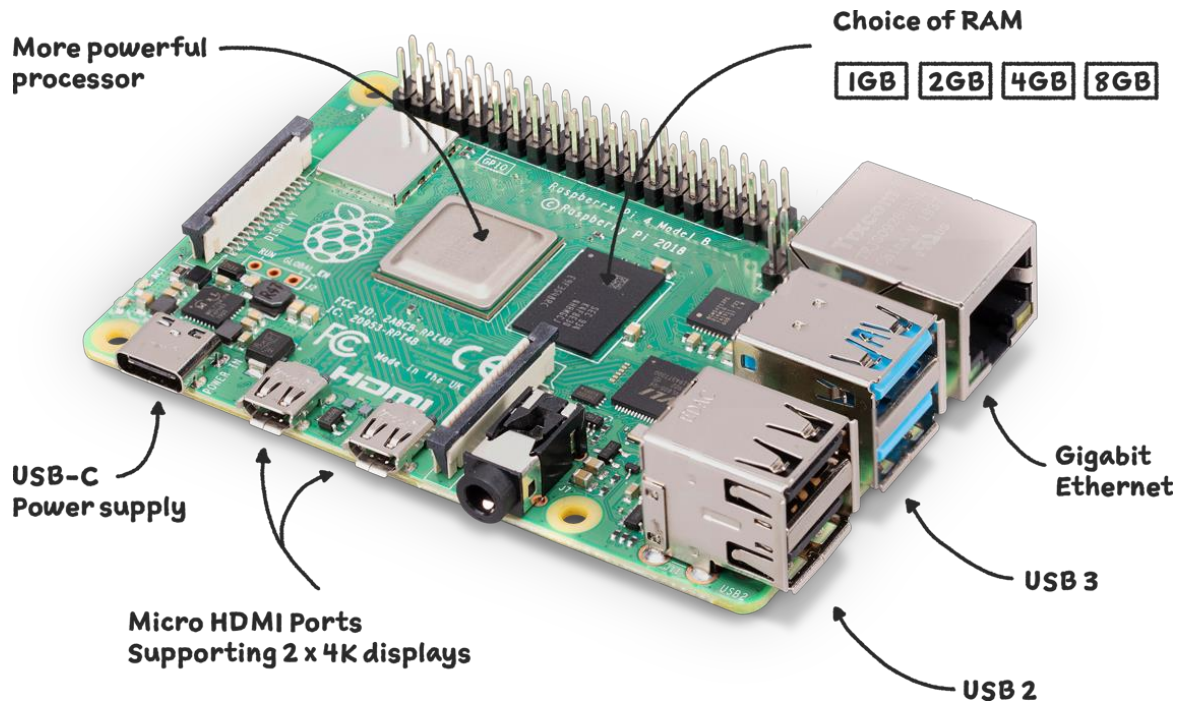


Momentary buttons: It is often referred to as push buttons, are electrical switches that remain in the "off" state until they are pressed. When pressed, they momentarily make an electrical connection, allowing current to flow temporarily. These buttons are commonly used in various electronic devices, such as keyboards and control panels, to trigger specific actions or functions when pressed and released.



Raspberry Pi 4 Model B

The Raspberry Pi 4 is a single-board computer developed by the Raspberry Pi Foundation. It is the fourth generation in the Raspberry Pi series, and it was released in June 2019. The Raspberry Pi 4 is a credit card-sized computer that is popular for its affordability, versatility, and the ability to run various operating systems.



3V3 power	1	2	5V power
GPIO 2 (SDA)	3	4	5V power
GPIO 3 (SCL)	5	6	Ground
GPIO 4 (GCLK0)	7	8	GPIO 14 (TXD)
Ground	9	10	GPIO 15 (RXD)
GPIO 17	11	12	GPIO 18 (PCM_CLK)
GPIO 27	13	14	Ground
GPIO 22	15	16	GPIO 23
3V3 power	17	18	GPIO 24
GPIO 10 (MOSI)	19	20	Ground
GPIO 9 (MISO)	21	22	GPIO 25
GPIO 11 (SCLK)	23	24	GPIO 8 (CE0)
Ground	25	26	GPIO 7 (CE1)
GPIO 0 (ID_SD)	27	28	GPIO 1 (ID_SC)
GPIO 5	29	30	Ground
GPIO 6	31	32	GPIO 12 (PWM0)
GPIO 13 (PWM1)	33	34	Ground
GPIO 19 (PCM_FS)	35	36	GPIO 16
GPIO 26	37	38	GPIO 20 (PCM_DIN)
Ground	39	40	GPIO 21 (PCM_DOUT)

We picked the Raspberry Pi 4 Model B because it's a strong computer with lots of ways to connect things to it, and many people can help us use it. This makes it a good choice for our device to help blind people. It can easily handle the job of detecting objects with YOLOv8 and using the TF-Luna LiDAR sensor. It's also small and doesn't use much power, so it's easy to carry around and will send audio messages to an earpiece to help blind people stay safe and independent.

Technical Details

Processor	Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
Memory	1GB, 2GB, 4GB or 8GB LPDDR4 (depending on model) with on-die ECC
Connectivity	2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, BLE Gigabit Ethernet 2 × USB 3.0 ports 2 × USB 2.0 ports
GPIO	Standard 40-pin GPIO header (fully backwards-compatible with previous boards)
Video & sound	2 × micro-HDMI ports (up to 4Kp60 supported) 2-lane MIPI DSI display port 2-lane MIPI CSI camera port 4-pole stereo audio and composite video port
Multimedia	H.265 (4Kp60 decode); H.264 (1080p60 decode, 1080p30 encode); OpenGL ES, 3.0 graphics
SD card support	Micro SD card slot for loading operating system and data storage
Input power	5V DC via USB-C connector (minimum 3A1) 5V DC via GPIO header (minimum 3A1) Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
Environment	Operating temperature 0–50°C

Raspberry Pi Camera Module 2

The Raspberry Pi Camera Module 2 has a Sony IMX219 8-megapixel sensor. It can be used to take high-definition video, as well as stills photographs.

It's a leap forward in image quality, colour fidelity, and low-light performance. It supports 1080p30, 720p60 and VGA90 video modes, as well as still capture. It attaches via a 15cm ribbon cable to the CSI port on the Raspberry Pi.



We chose the Raspberry Pi Camera Module v2 because it works well with the Raspberry Pi 4, has the right image quality, is small and affordable, and has good community support. These qualities made it the best fit for our blind assistance device.

We have significantly reduced the size of the footage we are capturing to only capture the objects that are directly in front of the device.

This also prevents the object detection model from reporting all the objects detected in one frame.



Technical Details

Size	Around 25 × 24 × 9 mm
Weight	3g
Still resolution	8 Megapixels
Video modes	1080p47, 1640 × 1232p41 and 640 × 480p206
Sensor	Sony IMX219
Sensor resolution	3280 × 2464 pixels
Sensor image area	3.68 x 2.76 mm (4.6 mm diagonal)
Pixel size	1.12 μm x 1.12 μm
Optical size	1/4"
Focus	Adjustable
Depth of field	Approx 10 cm to ∞
Focal length	3.04 mm
Horizontal Field of View (FoV)	62.2 degrees
Vertical Field of View (FoV)	48.8 degrees
Focal ratio (F-Stop)	F2.0
Maximum exposure times (seconds)	11.76

UPS HAT (D)

The UPS HAT (D) is a special board for Raspberry Pi that ensures power stays on. It has a built-in battery charger, boost converter, and voltage/current monitor. It can charge and provide steady 5V power. Battery information like voltage and capacity level can be checked through I2C. Plus, it shows the battery level on the Pi for easy monitoring.

Reasons for choosing UPS HAT (D) by WaveShare:

- **Reliability:** A UPS HAT ensures steady power for our device, which is crucial for helping visually impaired individuals, as it prevents data loss, safety risks, or unexpected stops when the power goes out.
- **Data Integrity:** The UPS HAT keeps the power stable, so our system can process data safely. This data includes things like object detection, distance measurements, and sound output. If the data is wrong or gets messed up, it can be dangerous for users and make the device less useful.
- **User Safety:** Keeping the device working all the time makes it safer for users. They depend on accurate information from the device for navigation and decisions, and a power problem could be bad.

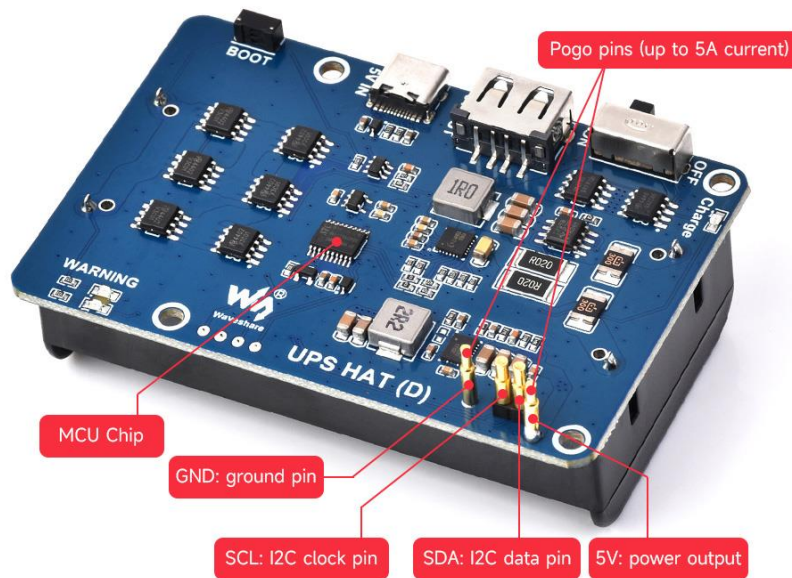
Technical Details

Output voltage	5V
Control bus	I2C
Battery support	21700 rechargeable Li battery 3.6V
Charger support	5V, Micro USB interface
Current capacity	5000mAh (Standard)
Dimensions	56 × 85mm
Mounting hole size	3.0mm

The UPS HAT (D) includes a USB Type C charging port with a 5V input voltage. LED indicators show charging status.

It has a power switch marked ON/OFF

Using the battery for power output requires activation of the battery protection chip by charging it or pressing the BOOT button.

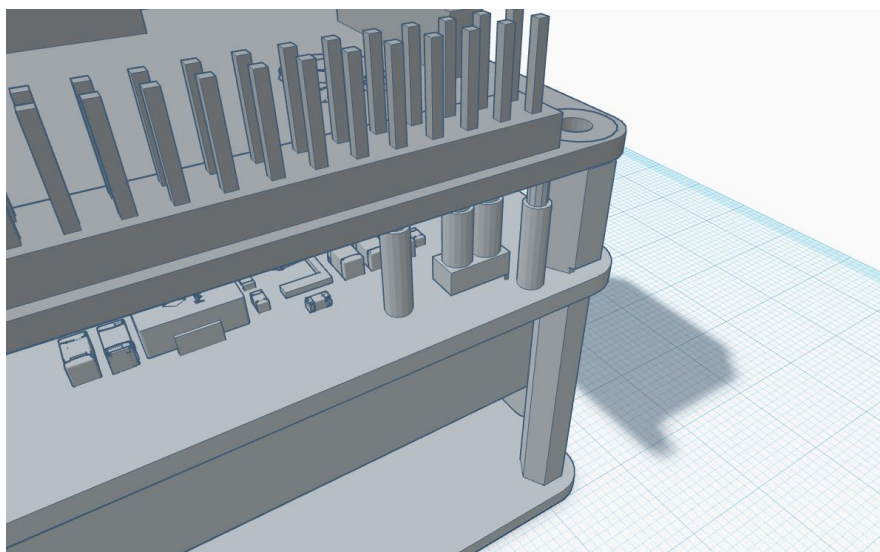


Safety Precautions

- Reversing the battery's negative and positive connection during charging or discharging should be avoided
- Batteries used should be compatible with Raspberry Pi 4

Hardware Setup

To correctly connect the UPS HAT (D) we just need to carefully mount the Pi on the UPS as shown.



The pogo pins connect to the GPIO Pin 2 (5V), Pin 3 and 5 (for I2C Communication) and Pin 9 (Ground)

Code

Step-1

First we need to enable I2C interface. Running the following command in terminal will open the configuration interface.

```
sudo raspi-config
```

Step-2

Next navigate to Interfacing Options, select I2C and enable the I2C kernel driver.

Step-3

Reboot the Pi:

```
sudo reboot
```

Step-4

Install 7zip and download the UPS HAT (D) demo files:

```
sudo apt-get install p7zip
wget https://files.waveshare.com/wiki/UPS-HAT-(D)/UPS_HAT_D.7z
7zr x UPS_HAT_D.7z -r -o./
cd UPS_HAT_D
```

Step-5

Run the demo using the following command:

```
python3 INA219.py
```

The demo will output battery voltage, current, power, and remaining battery capacity percentage.

Step-6

This will display the battery level icon in the upper right corner:

```
cd ~/UPS_HAT_D
./main.sh
sudo reboot
```

After rebooting, we should be able to see a battery icon with battery information.

When the battery level drops below 5%, a low battery warning will appear.

After 60 seconds, the Raspberry Pi will power off automatically. If you plug in the power supply during the warning, it will exit the warning interface.

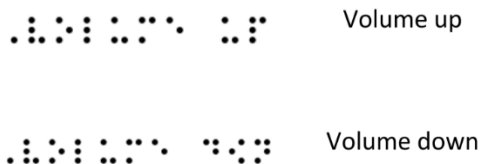
Volume Buttons

Code Link:

https://github.com/fall-blue/IRIS-Project/blob/main/IRIS_Volume_Control.py

(NOTE: Please avoid copying and pasting the code directly from this document, as it may cause formatting issues, please use the link for that)

We have incorporated two buttons with Braille engravings to facilitate volume control, enhancing accessibility for individuals with visual impairments.



Key Concepts

1. **GPIO:** GPIO stands for General Purpose Input/Output. It allows us to control and interact with external devices using digital signals.
2. **Volume Control:** Adjusting the volume level of audio output.

Code Structure

The code begins by importing the necessary libraries: RPi.GPIO and alsaaudio. The RPi.GPIO library provides functions to control the GPIO pins, while the alsaaudio library allows us to control the audio volume.

Next, we define the GPIO pins for volume control. We have two buttons: one for increasing the volume and another for decreasing the volume. The pins for these buttons are defined as VOLUME_UP_PIN and VOLUME_DOWN_PIN, respectively.

We then set up the GPIO mode and configure the pins as inputs with pull-up resistors. This ensures that the buttons are in the correct state when not pressed.

The code defines a constant VOLUME_STEP which represents the amount by which the volume will be increased or decreased each time the volume button is pressed. In this example, the volume is adjusted by **5 units**.

Next, we define two functions: increase_volume and decrease_volume. These functions are called when the volume buttons are pressed. The increase_volume function retrieves the current volume level, increases it by VOLUME_STEP, and sets the new volume level using the mixer.setvolume() function from the alsaaudio library. Similarly, the decrease_volume function decreases the volume level.

We use the `GPIO.add_event_detect()` function to detect button presses. When the volume up button is pressed, the `increase_volume` function is called. When the volume down button is pressed, the `decrease_volume` function is called.

Finally, we enter a loop that continuously runs until a keyboard interrupt (Ctrl+C) is detected. This loop ensures that the program remains active and can respond to button presses. The `GPIO.cleanup()` function is called to clean up the GPIO pins when the program is terminated.

Code Examples

```
import RPi.GPIO as GPIO
import alsaaudio

mixer = alsaaudio.Mixer()

# This sets the volume button pins
VOLUME_UP_PIN = 17
VOLUME_DOWN_PIN = 18

GPIO.setmode(GPIO.BCM)
GPIO.setup(VOLUME_UP_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(VOLUME_DOWN_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)

# Every time we press the volume button, the volume will be increased or
# decreased by 5
VOLUME_STEP = 5

# Function to increase volume
def increase_volume(channel):
    current_volume = mixer.getvolume()[0]
    new_volume = min(100, current_volume + VOLUME_STEP)
    mixer.setvolume(new_volume)

# Function to decrease volume
def decrease_volume(channel):
    current_volume = mixer.getvolume()[0]
    new_volume = max(0, current_volume - VOLUME_STEP)
    mixer.setvolume(new_volume)

GPIO.add_event_detect(VOLUME_UP_PIN, GPIO.FALLING, callback=increase_volume,
bouncetime=200)
GPIO.add_event_detect(VOLUME_DOWN_PIN, GPIO.FALLING, callback=decrease_volume,
bouncetime=200)

try:
    while True:
        pass
except KeyboardInterrupt:
    GPIO.cleanup()
```

To run this code, we must follow the following steps:

Step-1

Open terminal and install some essential programs by running the following command:

```
pip install RPi.GPIO  
pip install pyalsaaudio
```

Step-2

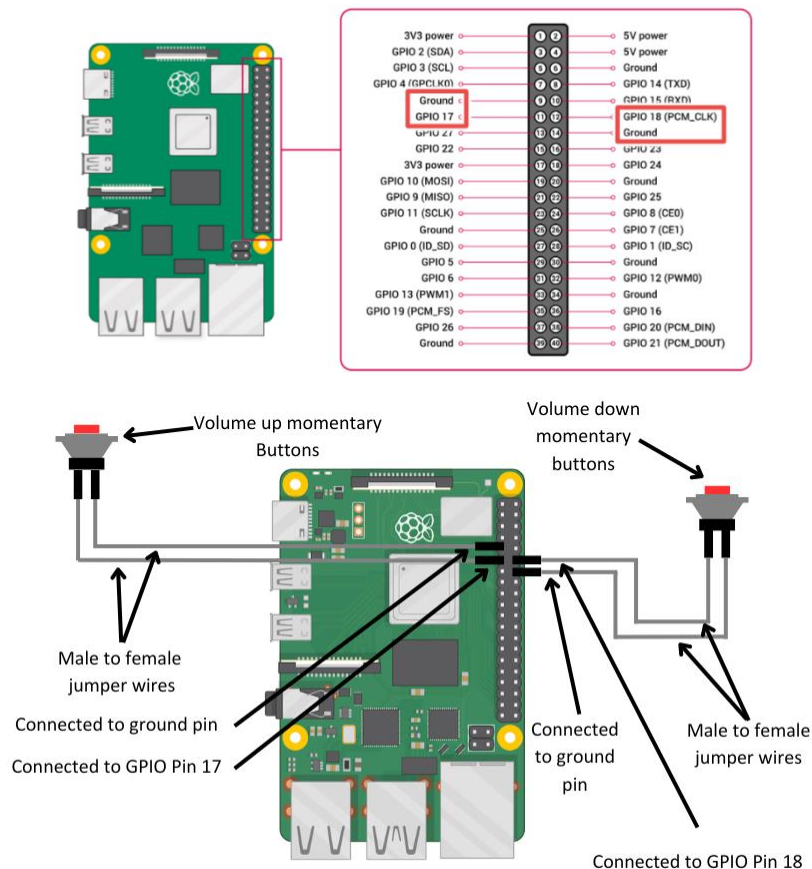
Save the above code as IRIS_Volume_Control.py

Step-3

Run the python script by running the following command in terminal.

```
python IRIS_Volume_Control.py
```

Hardware Setup



LiDAR Distance Sensor

LiDAR Sensors

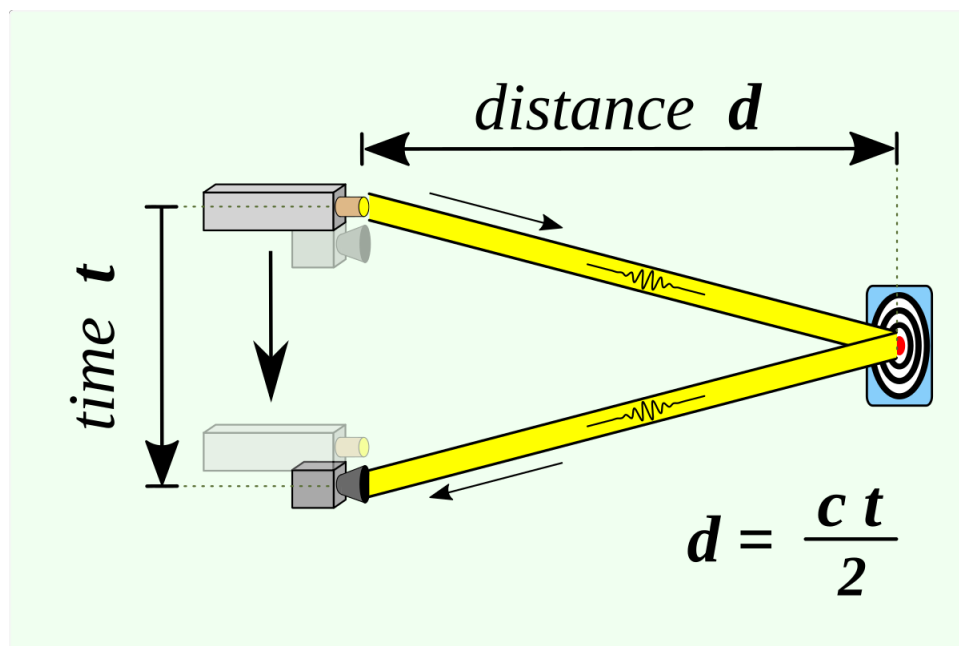
LiDAR stands for Light Detection and Ranging. It's a method of 3-D laser scanning.

Lidar sensors provide 3-D structural information about an environment. Advanced driving assistance systems (ADAS), robots, and unmanned aerial vehicles (UAVs) employ lidar sensors for accurate 3-D perception, navigation, and mapping.

Lidar is an active remote sensing system that uses laser light to measure the distance of the sensor from objects in a scene. A lidar sensor emits laser pulses that reflect off surrounding objects. The sensor then captures this reflected light and uses the time-of-flight principle to measure its distance from objects, enabling it to perceive the structure of its surroundings.

Time of flight principle

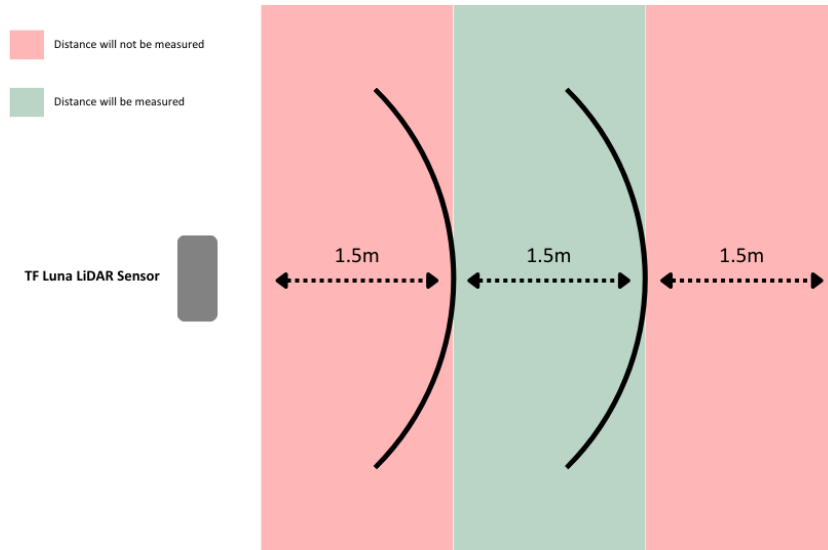
The Time-of-Flight principle (ToF) is a method for measuring the distance between a sensor and an object, based on the time difference between the emission of a signal and its return to the sensor, after being reflected by an object. Various types of signals (also called carriers) can be used with the Time-of-Flight principle, the most common being sound and light.



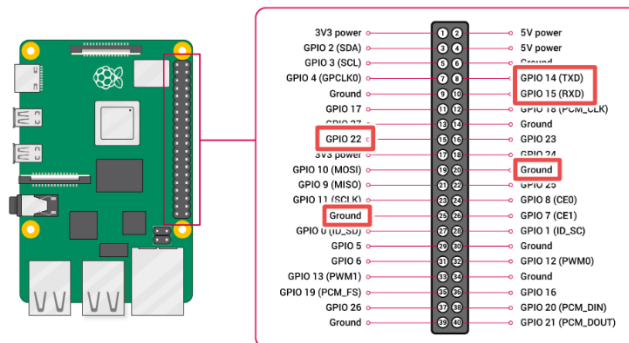
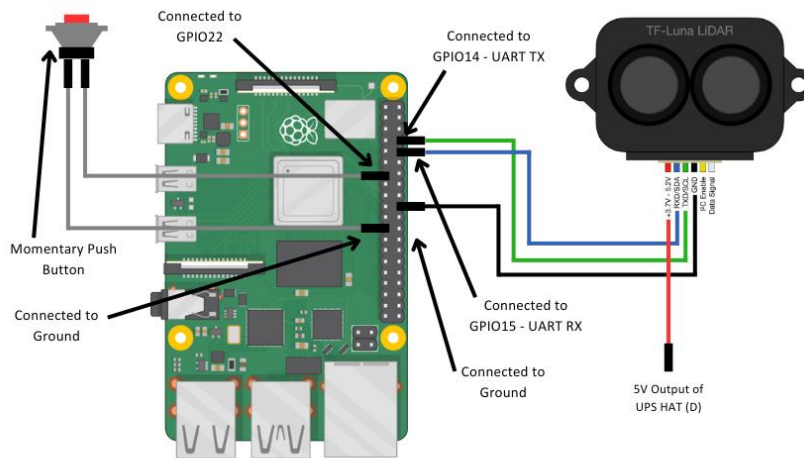
- d =distance
- c =speed of light
- t =time taken by the laser pulse to hit the object and return

In our device we have used TF-Luna Distance LiDAR Sensor which uses the ToF principle to measure the distance of the objects which are located between 1 and 2 meters of the device.

We decided to measure objects between 1.5 and 3 meters from the device. Things closer than 1.5 meter are already accessible to the blind person, and objects farther than 3 meters don't matter since the sensor will eventually measure them as the person moves forward.



Hardware Setup



Object Detection

Object detection is a computer vision technique for locating instances of objects in images or videos. Object detection algorithms typically leverage machine learning or deep learning to produce meaningful results.

Yolov8 is an object detection algorithm that aims to detect objects in images or video frames. Our model is trained to detect the following objects

- Chairs
- Tables
- Sofas
- Fireplaces

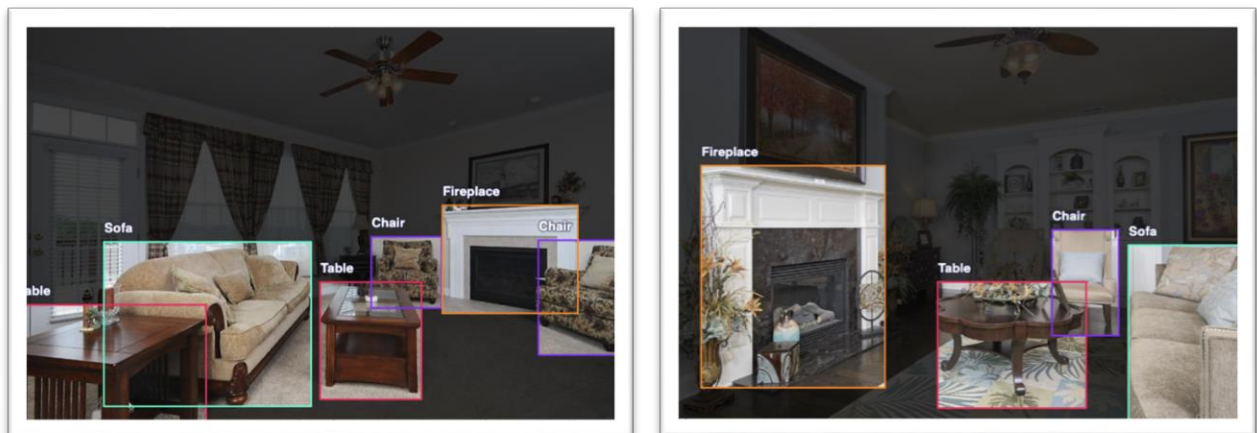
Architecture:

YOLOv8 short for "You Only Look Once version 8" is a state-of-the-art object detection model that belongs to the YOLO family of models. YOLOv8 is a deep neural network designed for real-time object detection in images and video frames. It is characterized by its efficient and streamlined architecture, making it suitable for applications where fast and accurate object detection is essential.

YOLOv8's architecture typically consists of numerous convolutional layers, down-sampling and up-sampling layers, and detection layers. These components work together to process input images and produce bounding boxes, class predictions, and confidence scores for detected objects.

Training Data

Our YOLOv8 Model is trained on a diverse dataset of 258 images containing labeled images with objects of interest. The training dataset is crucial for enabling the model to recognize a wide range of objects accurately.



Real-Time Detection

One of the distinguishing features of YOLOv8 is its ability to perform real-time object detection. Our model is optimized for speed without compromising accuracy. This real-time capability makes it suitable for applications where quick and responsive object detection is critical, such as in our assistive device for the blind.

Object Detection Process

The primary function of YOLOv8 is to detect objects in images or video frames. The detection process can be summarized as follows:

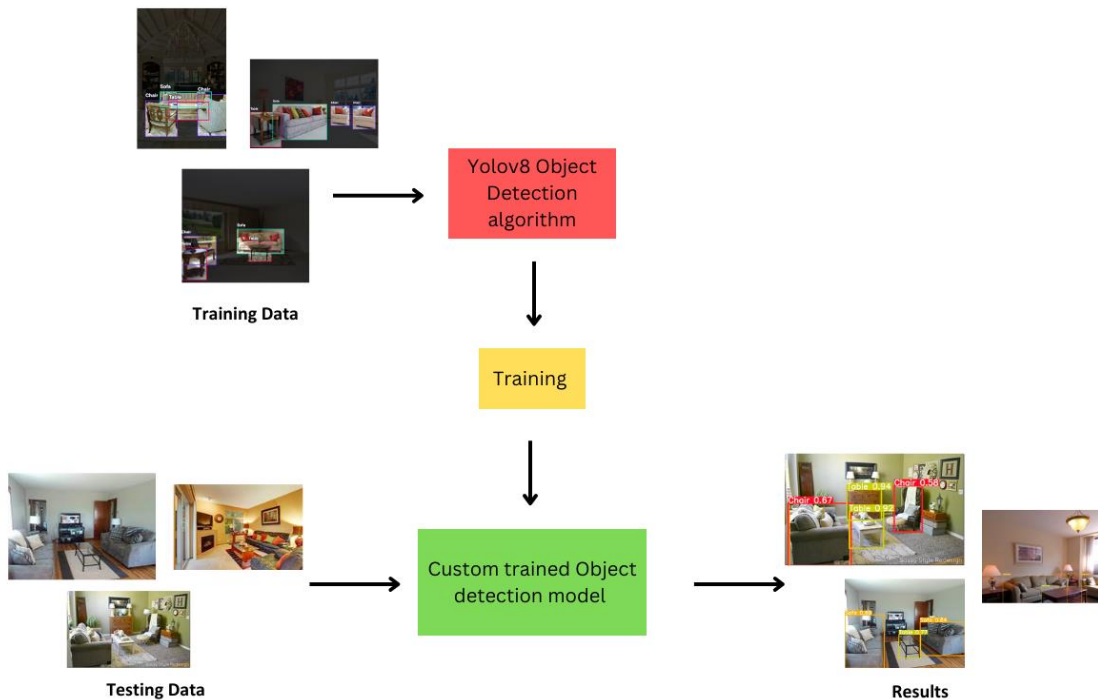
- **Input Data:** YOLOv8 takes an image or video frame as input.
- **Bounding Boxes:** The model processes the input and draws bounding boxes around objects it recognizes in the image. Each bounding box specifies the location and size of a detected object.
- **Confidence Scores:** YOLOv8 assigns a confidence score to each bounding box, indicating the model's level of confidence in the presence of an object. High confidence scores signify strong model certainty, while lower scores may imply uncertainty.
- **Class Labels:** In addition to confidence scores, YOLOv8 assigns class labels to the detected objects, specifying what each object is. For example, if the model identifies a chair, it will attach the class label "chair" to the corresponding bounding box.



(The numbers that you can see on top of the bounding box is the confidence score)

Performance and accuracy

YOLOv8 is known for its exceptional balance between speed and accuracy. Incorporating YOLOv8 into our assistive device for the visually impaired provides the capability to recognize objects and offer real-time feedback to users, enhancing their independence and safety. The efficient architecture and training approach of YOLOv8 make it an excellent choice for your project's object detection needs.



Code links:

This is the development code. In this colab file you can see how we developed the object detection model:

<https://colab.research.google.com/drive/1ZGJYXYCFE017xDfoDHoJFhx5euxfAbX3?usp=sharing>

In this colab file you can test our object detection model on your own dataset.

https://colab.research.google.com/drive/1epjUgjLYKvbi_0NmRiSrUMa7BkGdZwqR?usp=sharing

DISCLAIMER: this is just a prototype, and the model is still in development. We are actively working on expanding our dataset to incorporate more object classes, enhancing the model's versatility, performance and accuracy.

Object Detection with LiDAR Integration

Code link:

https://github.com/fall-blue/IRIS-Project/blob/main/IRIS_Lidar_Object_Detection.py

(NOTE: Please avoid copying and pasting the code directly from this document, as it may cause formatting issues, please use the link for that)

Introduction

This code implements an object detection system that integrates LiDAR (Light Detection and Ranging) technology. The system uses a Raspberry Pi, a LiDAR sensor, a camera, and a pre-trained object detection model to detect objects within a certain distance range and play an audio message indicating their presence.

Key Concepts

1. **Object Detection:** The process of identifying and localizing objects within an image or video.
2. **LiDAR:** A remote sensing technology that uses laser light to measure distances and create detailed 3D maps of the environment.
3. **Raspberry Pi:** A small, single-board computer that can be used for various projects and applications.
4. **gTTS:** A Python library that allows text-to-speech conversion using Google Text-to-Speech API.
5. **OpenCV:** A popular computer vision library that provides various functions for image and video processing.
6. **Torch:** A deep learning framework that provides tools for building and training neural networks.

Code Structure

The code is structured as follows:

1. Importing the necessary libraries and modules.
2. Loading the object detection model.
3. Initializing the LiDAR sensor and camera.
4. Defining the central region dimensions for object detection.
5. Implementing functions for collecting LiDAR data, detecting objects, and playing audio messages.
6. Setting up a loop to continuously monitor the button state and start/stop data collection.
7. Handling exceptions and cleaning up GPIO resources.

Code Examples

Here are some key code snippets from the provided code:

Loading the object detection model:

```
model = torch.load("")
```

This line loads our object detection model. The path needs to be specified in the `torch.load()` function.

Collecting LiDAR data:

```
def collect_data():
    global collecting_data
    try:
        while collecting_data:
            data = sensor_serial.readline().decode("utf-8").strip()
            if data:
                print("LiDAR Data:", data)
                distance = float(data) / 100 # this will convert LiDAR data to
meters
                detect_and_play_audio(distance)
    except Exception as e:
        print(f"Error in collect_data: {e}")
    finally:
        sensor_serial.close()
```

This function reads data from the LiDAR sensor and converts it to meters. It then calls the `detect_and_play_audio()` function to perform object detection and play audio messages.

Detecting objects and playing audio messages:

```
def detect_and_play_audio(distance):
    try:
        while True:
            ret, frame = camera.read() # this will read a frame from the camera

            cropped_frame = frame[:, central_x:central_x + central_width] # this
will crop the central region

            results = model(cropped_frame) # this will perform object detection
on the cropped frame

            for result in results.xyxy[0].cpu().numpy():
                class_id = int(result[5])
                class_label = model.names[class_id]
                if 1.5 <= distance <= 3.0: # this will filter objects within
1.5-3.0 meters
                    print(f"Detected {class_label} at {distance} meters")
                    play_audio_message(class_label, distance)
```



```

        cv2.imshow("Object Detection", cropped_frame) # this will display
the video frame
        key = cv2.waitKey(1) & 0xFF

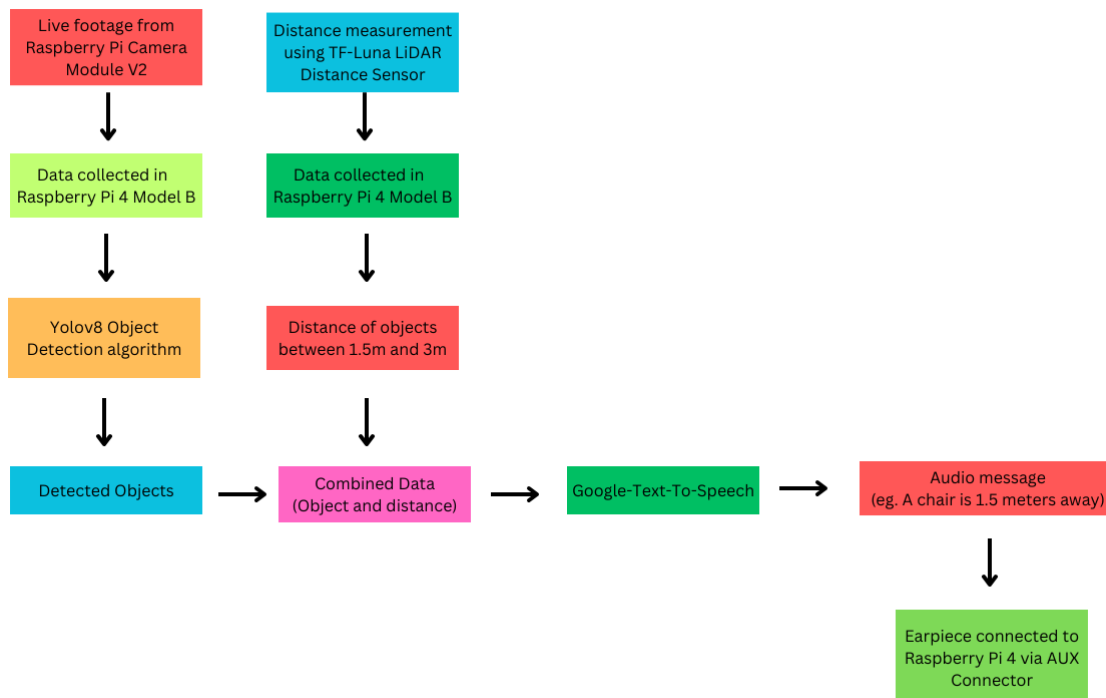
        if key == ord("q"):
            break
    except Exception as e:
        print(f"Error in detect_and_play_audio: {e}")
    finally:
        camera.release()
        cv2.destroyAllWindows()

```

This function reads frames from the camera, crops the central region, performs object detection using the pre-trained model, and plays audio messages for detected objects within the specified distance range.

Conclusion

This code demonstrates how to integrate LiDAR technology with object detection using a Raspberry Pi. By combining LiDAR data with visual information from a camera, the system can accurately detect objects within a certain distance range and provide audio feedback.



To run this code we must follow the following steps:

Step-1

Open terminal and run the following commands **individually** to install the required programs:

```
sudo apt-get update
sudo apt-get install python3
sudo apt-get install python3-pip
pip install gTTS opencv-python ultralytics
sudo apt-get install mpg123
pip install torch torchvision torchaudio
pip install pyserial
```

Step-2

Run the following command to install the object detection model.

```
git clone https://github.com/fall-blue/Living-Room-Object-Detection.git
```

Step-3

We need to enable UART interface for the lidar sensor.

(UART is a method for devices to talk to each other by sending data one piece at a time, like a conversation between a microcontroller or computer and other gadgets.)

Use the following command to access the configuration menu:

```
sudo raspi-config
```

Step-4

Under interfacing options enable the Serial interface and disable the login shell over serial.

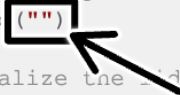
Step-5

Locate the object detection model. It will be saved as **Yolov8_ObjectDetection.pt**

Copy file path and paste it here:

```
# This will load the object detection model
model = torch.load("")

# this will initialize the lidar sensor
button_pin = 22
sensor_serial = serial.Serial('/dev/ttyS0', baudrate=115200)
GPIO.setmode(GPIO.BCM)
GPIO.setup(button_pin, GPIO.IN, pull_up_down=GPIO.P
```



Paste the file path here

Step-6

After pasting the file path in between the inverted commas. Save the main code as IRIS_Lidar_Object_Detection.py

Step-7

Run the following command in terminal.

```
python IRIS_Lidar_Object_Detection.py
```

Running IRIS_Lidar_Object_Detection.py on Raspberry Pi 4 bootup

This code will automatically run the python script and start detection and distance measurement when the Raspberry Pi 4 turns on.

(NOTE: Please avoid copying and pasting the code directly from this document, as it may cause formatting issues)

Step-1

Open terminal and run the following command. This will create a service unit file.

```
sudo nano /etc/systemd/system/IRIS_Lidar_Object_Detection.service
```

Step-2

Add the following data to the service file that was just created. Then save and exit the file.

(NOTE: make sure to write the file path in ExecStart and folder path in Working directory)

```
[Unit]
Description=Object Detection and Lidar Data Collection

[Service]
ExecStart=/usr/bin/python3 /replace this with the exact path of
IRIS_Lidar_Object_Detection.py
WorkingDirectory=/replace this with the exact path of the folder where
IRIS_Lidar_Object_Detection.py is saved
Restart=always
User=pi

[Install]
WantedBy=multi-user.target
```

Step-3

This will enable the service when the Raspberry Pi 4 turns on.

```
sudo systemctl enable IRIS_Lidar_Object_Detection.service
```

Step-4

Running this command will start the service.

```
sudo systemctl start IRIS_Lidar_Object_Detection.service
```

Step-5

To check the status of the service run the following command. If there are any errors, then we will be able view them here as it will show us the output generated by

IRIS_Lidar_Object_Detection.py

```
sudo systemctl status IRIS_Lidar_Object_Detection.service
```

Step-6

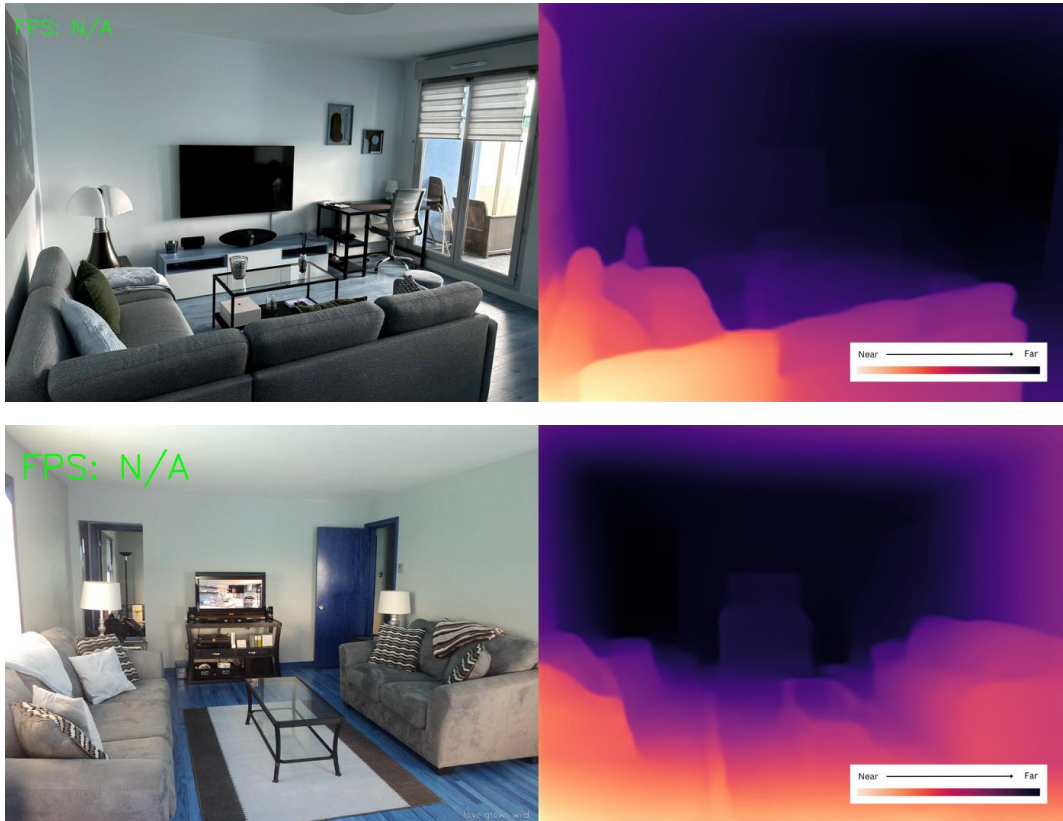
To test the service, reboot the Raspberry Pi. You can run the following command to reboot.

```
sudo reboot
```

Depth Estimation MiDaS Neural Network

MiDaS (Multiple Depth Estimation Accuracy with Single Network) is a deep learning based residual model built atop Res-Net for monocular depth estimation. MiDaS is known to have shown promising results in depth estimation from single images.

Examples:



Code link:

In this colab file you can test your own images and use estimate depth:

https://colab.research.google.com/drive/1kavGULJsKu3lzpxc6mA0FdAZ_F3cWabF?usp=sharing

Architecture

MiDaS uses two main parts: the encoder extracts important information from the input, and the decoder creates a depth map using this information.

Backbone

MiDaS uses a robust network like ResNet-50 or ResNet-101 to capture different details in the input images at various scales, which helps create depth maps effectively.

Multi-Scale Feature Fusion

In MiDaS, they do a few clever things to make sure they get depth information from pictures as accurately as possible.

First, they combine different pieces of information from the picture taken at different distances to get a better overall idea of how far things are from the camera. This helps make the depth estimation more precise.

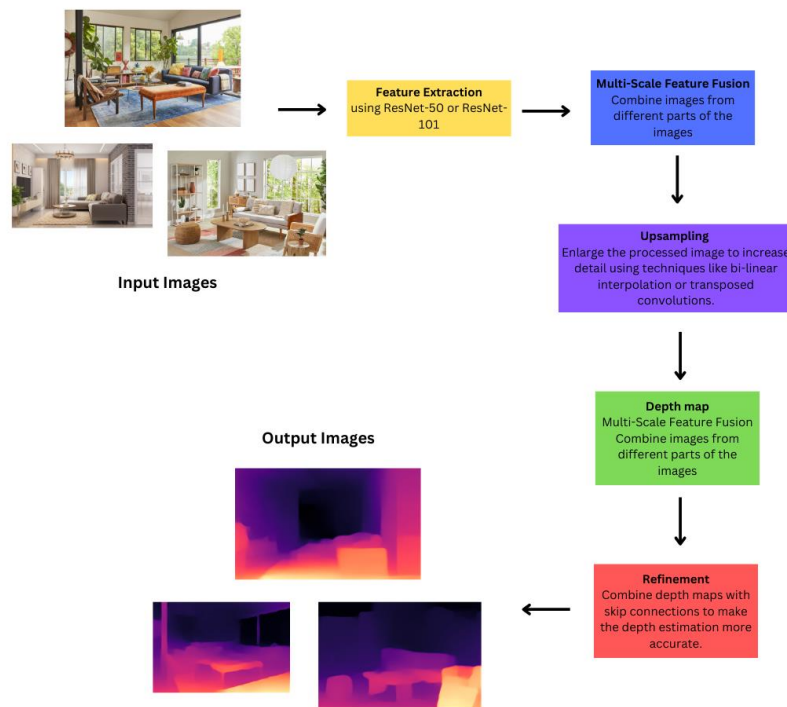
Second, they connect the parts of the picture with a lot of detail (like fine lines and textures) to the parts that look more general. This way, the model can use both the fine details and the bigger picture to figure out depth.

Finally, they take all these combined pieces of information and put them together to make a better depth map. This map helps understand how far away things are in the picture. All of these techniques make the depth estimation more accurate.

Up-Sampling and Refinement

To create the depth map, MiDaS makes the picture bigger and more detailed. They do this using methods like stretching the picture or adding extra information to it. This makes the depth estimation more precise.

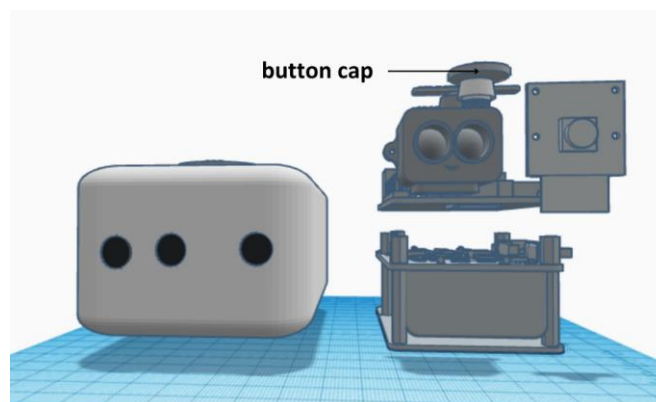
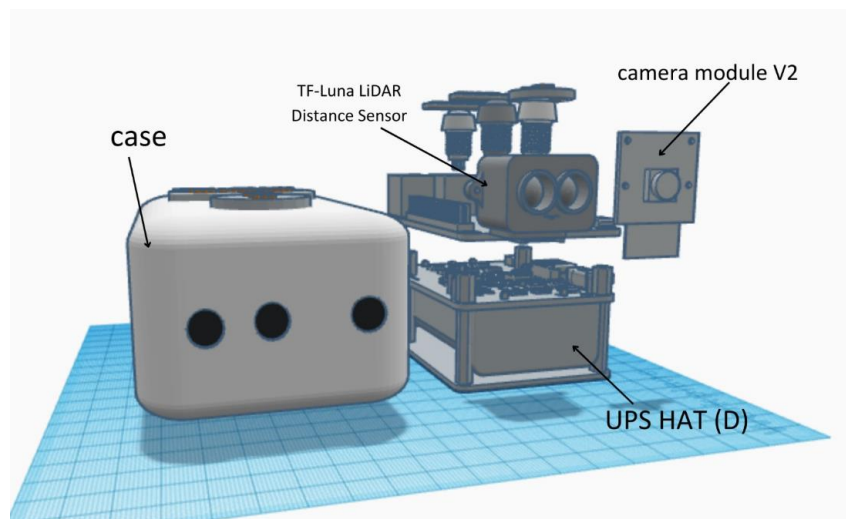
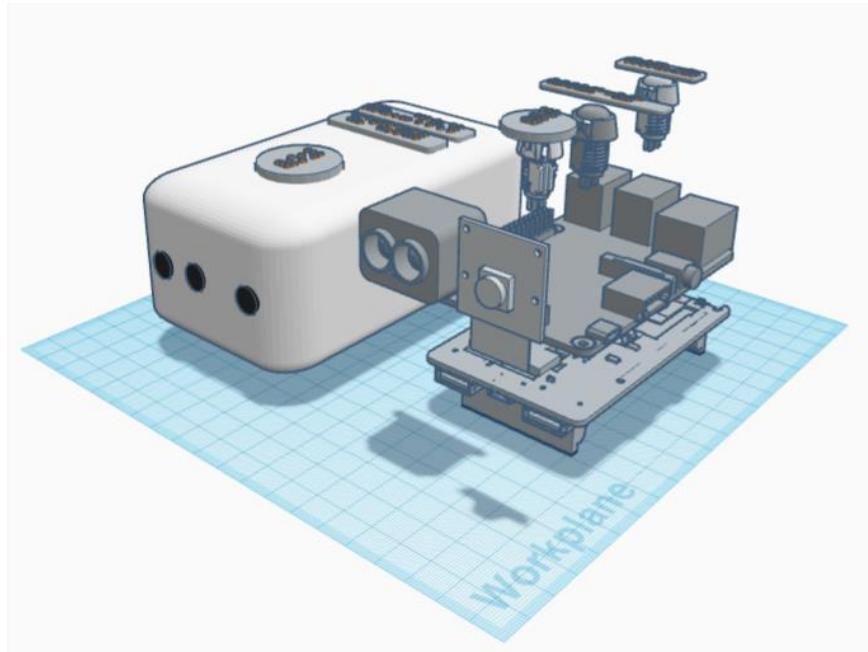
Then, they take all the different depth maps and combine them with related connections in the picture. This helps improve the accuracy of the depth estimation.

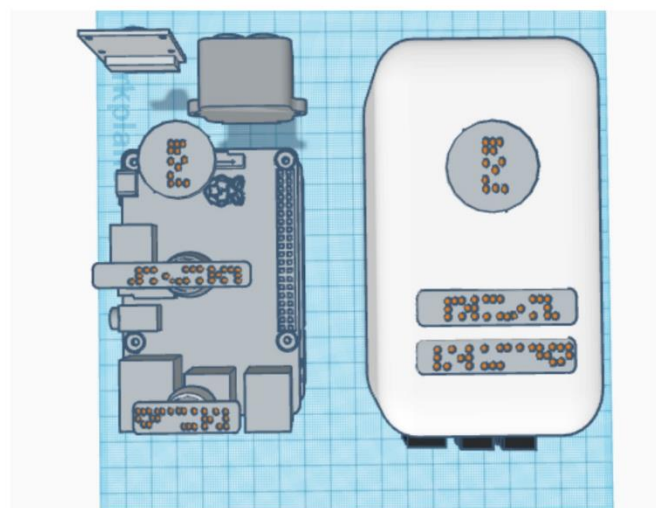
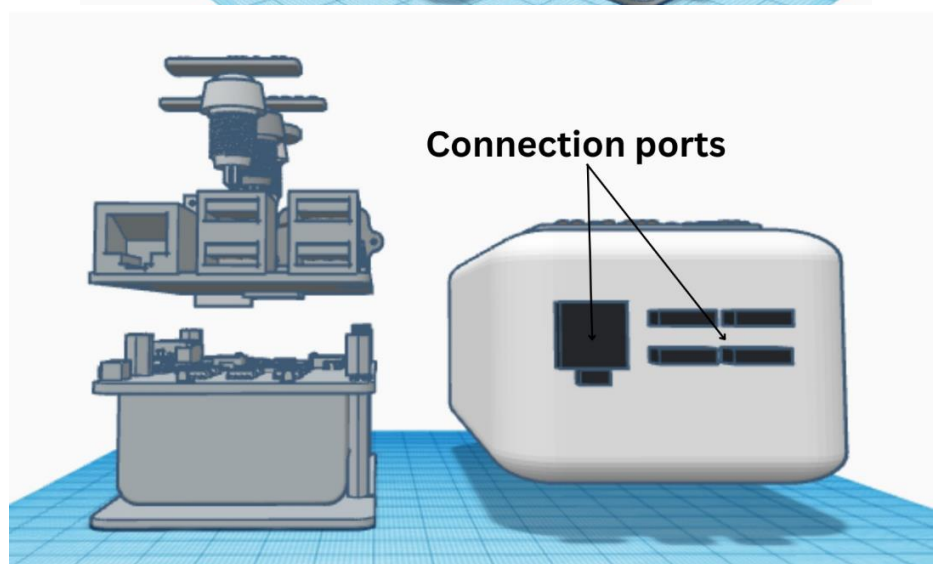
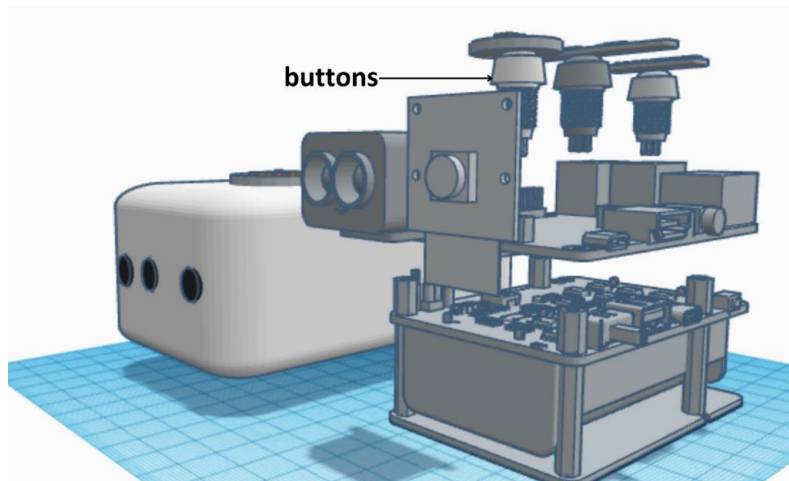


Originally, our plan was to use this Midas depth estimation model to measure object distances detected by our YOLOv8 system on a Raspberry Pi 4 camera in real-time. However, due to the high computational requirements and lack of suitable hardware, we've decided not to implement this at the moment. Instead, we've opted for LIDAR sensors as a more feasible solution. We still decided to share this information because we believe that in the future, integrating the Midas model alongside LIDAR data could significantly enhance accuracy.

3D Model

Link: <https://autode.sk/3Qg4uBW>





Conclusion

We've successfully developed a device to assist blind individuals, and we're excited about its potential to make a positive impact on their lives. Our creation centers around a Raspberry Pi 4 and integrates cutting-edge technologies to enhance mobility and awareness for visually impaired individuals.

First and foremost, we've implemented YOLOv8, a state-of-the-art object detection model. This remarkable technology enables us to recognize everyday objects such as chairs, sofas, and tables, providing critical information about the environment in real time.

To ensure our users' safety and confidence, we've integrated a TF Luna LiDAR distance sensor. This sensor accurately measures the distance to detected objects within a range of 1 to 2 meters, giving our users precise spatial information to navigate safely and confidently.

But we didn't stop there; we wanted to make this information accessible in the most user-friendly way. That's why we've incorporated Google Text-to-Speech to convert the data into audio messages. These messages are sent to an earpiece through an aux connector, allowing our users to receive immediate, verbal feedback about their surroundings.

In summary, our device is a powerful combination of advanced object detection, precise distance sensing, and clear, real-time audio feedback. We're incredibly proud of this creation, and we believe it has the potential to greatly improve the independence and safety of visually impaired individuals as they navigate their world.

Cost:

Device	Quantity	Cost (INR)	Link
Raspberry pi 4 model B	1	5,783	https://amzn.eu/d/apDu8Ko
Raspberry pi camera module V2	1	2,060	https://amzn.eu/d/apDu8Ko
TF-Luna LiDAR Distance Sensor	1	2000	https://robu.in/product/tf-luna-lidar-distance-sensor/
The UPS HAT (D)	1	3800	https://shorturl.at/hpFHM
Momentary buttons	3	30	https://shorturl.at/hCIKT
3D Printed Case	1	Unknown at the moment	
TOTAL	-	13673	-

Future Work:

1. We will make a real-life prototype.
2. We will work on expanding our dataset to incorporate more object classes, enhancing the model's versatility, performance and accuracy.
3. Add more sensors like ultrasonic sensors to improve object detection and distance measurement.
4. Improve the overall device in all aspects
5. Make accessories for this device so that it can be mounted on wheelchairs, sticks, shoulders etc.

References

<https://roboticsbackend.com/make-a-raspberry-pi-3-program-start-on-boot/>

<https://www.raspberrypi.com/>

[https://www.waveshare.com/wiki/UPS_HAT_\(D\)](https://www.waveshare.com/wiki/UPS_HAT_(D))

<https://www.terabee.com/time-of-flight-principle/>

<https://www.mathworks.com/help/lidar/ug/lidar-processing-overview.html>

<https://www.youtube.com/watch?v=p86AbggRalo&t=148s>

<https://en.benewake.com/DataDownload/index.aspx?pid=20&lcid=21>

<https://www.youtube.com/>